МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторної роботи 4
за темою «Використання масивів і вказівників»
з курсу «Алгоритмізація та програмування. Частина 1»

для студентів спеціальності
122 «Комп’ютерні науки»

Харків 2019

Методичні вказівки до виконання лабораторної роботи 4 за темою «Використання масивів і вказівників» з курсу «Алгоритмізація та програмування. Частина 1» для студентів спеціальності 122 «Комп'ютерні науки» / уклад. Л. В. Іванов, М. О. Білова. – Харків : НТУ «ХПІ», 2019. – 29 с.

Укладачі: Л. В. Іванов
М. О. Білова

Рецензент О. В. Шматко

Кафедра програмної інженерії та інформаційних технологій управління

ЗМІСТ

Вступ

Курс «Алгоритмізація та програмування» присвячений теоретичним та практичним аспектам розробки алгоритмів і програм мовою С++. Отримані в результаті вивчення даної дисципліни знання та навички можуть бути використані в усіх наступних курсах, під час виконання курсових та дипломних проектів, вивчення дисциплін, що пов'язані з обчислювальною технікою та програмуванням.

У процесі виконання лабораторної роботи за темою «Використання масивів і вказівників» студенти мають закріпити теоретичний матеріал, отримати навички практичної роботи при реалізації програм, пов'язаних з сортуванням та використанням масивів у динамічній пам'яті.

Перша частина присвячена теоретичним аспектам, зокрема використанню побітових операцій, визначенню та використанню одновимірних та багатовимірних масивів, визначенню та використанню вказівників. Проаналізовано зв'язок масивів зі вказівниками, надано рекомендації стосовно використання динамічної пам'яті. У практичній частині подано код для семи програм, у тому числі для сортування елементів масиву і створення двовимірного масиву у динамічній пам'яті.

Друга частина складається з завдань для лабораторної роботи, що використовуються як поточний контроль засвоєння матеріалу. Лабораторні заняття розраховані на роботу в комп'ютерному класі під безпосереднім керівництвом викладача та самостійну роботу студентів, яка передбачає написання тексту програми, закріплення практичних навичок роботи на комп'ютері, набутих при виконанні відповідної лабораторної роботи. Варіанти індивідуальних завдань видаються викладачем на занятті.

У методичних вказівках подано п'ять завдань. У цілому методичні вказівки будуть корисні студентам різних спеціальностей і форм навчання, які вивчають мову С ++.

ТЕОРЕТИЧНА ЧАСТИНА

1. Використання побітових операцій

C++ надає можливість використання так званих побітових операцій, які працюють з окремими бітами цілих чисел. Операнди повинні бути цілими числами (переважно цілі без знака). Наприклад:

```
unsigned char c1 = 168;      // 10101000
unsigned char c2 =  26;      // 00011010
```

Операція ТА (&, один амперсанд, на відміну від логічної операції), якщо її застосувати до двох бітів, повертає 1, якщо обидва біти мають значення 1, та 0 в іншому випадку:

```
unsigned char c3 = c1 & c2; // 00001000, або 8
```

Операція АБО (|, одна вертикальна риска, на відміну від логічного АБО), якщо її застосувати до двох бітів, повертає 0, якщо обидва біти мають значення 0, та 1 в іншому випадку:

```
unsigned char c4 = c1 | c2; // 10111010, або 186
```

Операція ВИКЛЮЧНЕ АБО (^) для двох однакових бітів встановлює результат в одиницю, якщо біти різні та нуль в іншому випадку:

```
unsigned char c5 = c1 ^ c2; // 10110010, або 178
```

Операція побітового заперечення або доповнення (~) замінює нулі на одиниці, а одиниці на нулі. Наприклад:

```
unsigned char c6 = ~c1;      // 01010111, або 87
```

Оператор << переміщує біти ліворуч. Ця операція відкидає крайній лівий біт і привласнює 0 крайньому правому біту. Наприклад:

```
unsigned char c7 = c1 << 1;  // 01010000, або 80
```

Оператор >> переміщує біти праворуч. Ця операція відкидає крайній правий біт і привласнює 0 крайньому лівому біту. Наприклад:

```
unsigned char c8 = c1 >> 2; // 00101010, або 42
```

Можна використовувати складене присвоєння: <<= і >>=.

2. Визначення одновимірних масивів

Масив – це структура даних, що складається з елементів одного типу. Усі елементи масиву займають суцільний блок пам'яті.

Масив можна створити, вказавши після змінної розмір масиву в квадратних дужках. Наприклад:

```
long longArray[25];
```

У такий спосіб визначено масив 25 цілих типу `long int`. Цей масив має ім'я `longArray`. У цьому прикладі компілятор виділяє блок пам'яті, в якому можна розташувати 25 довгих цілих значень. Оскільки кожне довге ціле займає 4 байта, компілятор резервує 100 послідовних байтів пам'яті.

Для визначення розміру масиву можна вживати лише константи – явні або іменовані, тобто такі, що може обчислювати компілятор, а також константні вирази. Не можна використовувати константи, які неможливо обчислити під час компіляції, а також змінні:

```
const int n = 10;       // явна константа
int firstArr[n];        // OK
```

```
    const int n1 = n + 3;  // константний вираз
    int secondArr[n1 + 2]; // OK: розмір може бути обчислений
компілятором
    // Наступна константа не може бути обчислена компілятором:
    const int m = std::pow(3, 3) / 3;
    int thirdArr[m];       // помилка компіляції
    int m1 = 4;            // змінна
    int fourthArr[m1];     // помилка компіляції
```

Елемент масиву можна отримати через його індекс (номер елемента). Нумерація елементів починається з 0. До першого елемента можна звернутися так: longArray[0]. Останній індекс повинен буди на одиницю менше кількості елементів. Для описаного вище масиву longArray – це 24.

Мова C++ не підтримує автоматичного контролю виходу значення індексу за межі масиву. Програміст сам повинен забезпечити цей контроль.

Під час створення масиву його можна ініціалізувати, тобто присвоїти початкові значення елементам. Список початкових значень вказують у фігурних дужках. Наприклад,

```
int integerArray[5] = {1, 2, 3, 4, 5};
```

Ми визначили масив `integerArray` з п’яти цілих. Відповідні початкові значення присвоюються елементам масиву. Можна опустити розмір масиву. Тоді компілятор визначить кількість елементів зі списку початкових значень. Попереднє визначення можна було б переписати так:

```
int integerArray[] = {1, 2, 3, 4, 5};
```

Не можна ініціалізувати більше елементів, ніж оголошено під час опису масиву. Тому наведений нижче запис

```
int integerArray1[5] = {1, 2, 3, 4, 5, 6};
```

призведе до помилки компіляції. Але, навпаки, можна написати так:

```
int integerArray2[5] = {1, 2};
```

Решта елементів буде ініціалізована значеннями 0.

3. Використання одновимірних масивів

Для отримання окремого елемента масиву використовують операцію отримання індексу (`[]`). Значення індексу вказують за допомогою константи, змінної або виразу, якщо їх можна привести до цілого типу:

```
int i = 0;
integerArray[i] = 11;
k = integerArray[2];
integerArray[k % 10 + 1] = 0;
```

Не можна присвоїти один масив іншому. Це синтаксична помилка. Для копіювання елементів одного масиву в інший слід створити окремий цикл:

```
const int n = 4;
double a[n];
double b[n] = {1, 2, 4, 8};
for (int i = 0; i < n; i++)
{
    a[i] = b[i];
}
```

Імена масивів можуть бути використані як фактичні параметри функції. Ім’я масиву без квадратних дужок є адресою першого елемента масиву. Копія цієї адреси присвоюється відповідному параметру. Функція може використовувати цю адресу для доступу до елементів масиву. Зміни, внесені в елементи, залишаються після завершення функції. Наприклад:

```
#include <iostream>
using namespace std;
```

```
void f(double a[])
{
    a[0] = 0;
}

int main()
{
    const int n = 4;
    double a[] = {1, 2, 4, 8};
    int i;
    for (i = 0; i < n; i++)
    {
        cout << a[i] << ' '; //1 2 4 8
    }
    cout << endl;
    f(a);
    for (i = 0; i < n; i++)
    {
        cout << a[i] << ' '; //0 2 4 8
    }
    cout << endl;
    return 0;
}
```

Не можна створити масив посилань, але можна визначити посилання на певний елемент масиву:

```
double a[] = {1, 2, 4, 8};
double& y = a[3];
y = 16; // a[3] = 16;
```

4. Багатовимірні масиви

Мова C++ дозволяє створювати масиви з більш ніж одним виміром. Кожен вимір подано як окремий індекс масиву. Масиви можуть мати будь-яку кількість вимірів, однак найчастіше створюють одновимірні і двовимірні масиви. Двовимірний масив можна подати як прямокутну таблицю. Тоді перший індекс найчастіше пов’язують з номером рядку, а другий – з номером стовпця.

Багатовимірні масиви також можна ініціалізувати. Наприклад:

```
    int theArray[5][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15 };
```

Це означає, що перші три елементи записують у рядок з індексом 0, наступні 3 – у рядок з індексом 1 тощо.

Для більшої наочності елементи можна згрупувати вкладеними дужками. Наприклад,

```
int theArray[5][3] = {{1,  2,  3},
                      {4,  5,  6},
                      {7,  8,  9},
                      {10, 11, 12},
                      {13, 14, 15}};
```

Компілятор ігнорує внутрішні фігурні дужки. Їх використовують тільки для того, щоб краще зрозуміти, як значення розділені. Значення повинні бути розділені комою, без урахування дужок. Весь набір ініціалізації повинен бути взятий у фігурні дужки і повинен закінчуватися крапкою з комою.

Тільки перша пара дужок може бути порожньою:

```
int theArray[][3] = {{1,  2,  3},
                     {4,  5,  6},
                     {7,  8,  9},
                     {10, 11, 12},
                     {13, 14, 15}};
```

У цьому випадку компілятор обчислює перший розмір автоматично.

5. Визначення вказівників

Кожна змінна знаходиться в унікальному місці у пам’яті комп’ютера. і має свою адресу. **Вказівник** є змінною, яка містить адресу пам’яті. Для визначення вказівника використовують зірочку, яка передує імені змінної. У наступному прикладі змінна `p` може зберігати адресу цілої змінної:

```
int *p;
```

Вказівник, значення якого дорівнює нулю, має назву нульового вказівника. Усі вказівники, коли вони створюються, одразу або пізніше повинні бути ініціалізовані. Якщо певне значення поки не можна визначити, вказівнику доцільно присвоїти нуль. Вказівник, який не ініціалізовано, називається **диким вказівником** (wild pointer). Дикі вказівники дуже небезпечні.

Вказівник можна ініціалізувати адресою іншої змінної, до якої застосовують операцію отримання адреси (&). Отримання адреси також використовують для присвоєння значення існуючому вказівнику. Наприклад:

```
int i = 5;
int *p = &i;
```

Щоб отримати доступ до даних, що зберігаються за адресою, можна використовувати так зване **розіменування** (dereferencing). Для розіменування вказівника оператор розіменування (*) встановлюється перед ім'ям вказівника. Наприклад:

```
int j = *p + i; // j = 5 + 5 = 10
```

Фактично `*p` є синонімом `i`. З `*p` можна працювати як зі звичайною цілою змінною.

```
*p = 6; // i = 6
```

Можна створити масив указівників:

```
int i = 0;
int j = 1;
int k = 10;
int *pa[3];
pa[0] = &i;
```

```
pa[1] = &j;
pa[2] = &k;
for (int i = 0; i < 3; i++)
{
    cout << *pa[i] << ' '; // 0 1 10
}
```

Можна здійснити ініціалізацію масиву вказівників адресами змінних:

```
int *pb[] = {&i, &j, &k};
for (int i = 0; i < 3; i++)
{
    cout << *pb[i] << ' '; // 0 1 10
}
```

Правила сумісності типів для вказівників є більш жорсткими, ніж аналогічні правила для значень. Можна присвоїти ціле значення дійсному і навпаки, але не можна присвоювати вказівники різних типів один одному:

```
int k = 1;
double d = k;
int j;
j = d;
int *pk = &k;
double *pd;
int *pj;
pj = pk;            // OK. Два вказівника вказують на одну змінну
pd = pk;         // Синтаксична помилка
pi = pd;         // Синтаксична помилка
float *pf = pd; // Синтаксична помилка
```

Існує особливий тип вказівників: `void*`. Можна призначити вказівники будь-яких типів до `void*`. Змінні типу `void*` використовуються для зберігання вказівників різних типів. У наступному прикладі масив `void*` містить адреси різних змінних:

```
int i = 0;
double d = 1.5;
char c = 'A';
```

```
void *p[4] = {&i, &d, &c}; // p[3] = 0
double *pd = &d;
p[3] = pd;
```

Не можна розіменовувати `void*`. Для того щоб отримати вказівники конкретних типів, слід привести вказівники до необхідних типів за допомогою оператора `static_cast`:

```
char *pc = static_cast<char*>(p[2]);
cout << *PC; // 'A'
```

Можна створювати вказівники на константний об'єкт: значення, на яке вказує вказівник не може бути змінено, але сам вказівник може бути переміщений на іншу змінну або константу.

```
int k = 4;
const int *pk = &k; // Вказівник на константу
k = 5;
cout << *pk;        // 5
*pk = 6;            // Помилка!
```

Константний вказівник (ключове слово const стоїть після *) – це вказівник, який не може бути змінений. Проте значення, на яке вказує вказівник, може бути змінено. Константні вказівники обов'язково повинні бути ініціалізовані:

```
int i = 1;
int * const cp = &i;
cout << *cp;         // 1
int j = 2;
cp = &j;             // Помилка!
```

Можна також створювати константні вказівники на константні об'єкти.

Аргументи функції можуть бути оголошені як вказівники. Це дозволяє надсилати адреси заданих змінних у функцію і змінювати ці змінні,

використовуючи їх адреси. Припустимо, що нам потрібно поміняти місцями значення двох цілих змінних. Раніше розглядався приклад функції з використанням посилань. Функція може також використовувати вказівники як параметри:

```
void swap(int *p1, int *p2)
{
    int x = *p1;
    *p1 = *p2;
    *p2 = x;
}
```

Тепер функцію `swap()` можна викликати з функції `main()`:

```
void main()
{
    int a = 1;
    int b = 2;
    swap(&a, &b); // Отримуємо адреси змінних
    cout << a << endl; // 2
    cout << b << endl; // 1
}
```

6. Зв'язок масивів зі вказівниками

У C++ ім'я масиву є постійним указівником на перший елемент масиву. Таким чином, у визначенні

```
int a[50];
```

`a` – це вказівник на `&a[0]`, тобто на початковий елемент масиву. Можна використовувати імена масивів як константні вказівники і навпаки:

```
int a[5] = {1, 2, 4, 8, 16};
int *p = a;
cout << *a << endl;   // 1
cout << p[2] << endl; // 4
```

До вказівників можна застосувати деякі операції. Арифметика вказівників обмежується додаванням, відніманням і порівнянням. Під час виконання арифметичних операцій з указівниками передбачається, що вказівники вказують на масив об'єктів. Додавши ціле значення до вказівника, ми переміщуємо його на відповідне число об'єктів у масиві. Якщо, наприклад, тип має розмір 10 байт, а потім ми додали ціле число 5, вказівник переміщується на 50 байт у пам'яті. Можна застосувати інкремент і декремент, а також операції складеного присвоювання для неконстантних указівників.

```
int a[5] = { 1, 2, 4, 8, 16 };
int *p = a;                 // p вказує на a[0]
cout << *(a + 3) << endl; // 8
p++;                        // p вказує на a[1]
cout << p << endl;          // 2
p += 3;                     // p вказує на a[1]
cout << p << endl;          // 16
a++;                         //  Помилка!  a  є  константним
вказівником
```

Різниця між двома вказівниками на різні елементи масиву повертає кількість елементів, які розташовані між цими вказівниками (включаючи перший і не включаючи останній). Наприклад:

```
int a[5] = { 1, 2, 4, 8, 16 };
int *p1 = a;                // p1 вказує на a[0]
int *p2 = a + 3;            // p2 вказує на a[3]
cout << p2 - p1             // 3;
```

Різниця між двома вказівниками має сенс тільки тоді, коли обидва вказівники вказують елементи одного масиву.

Перевірка виходу за межі масиву не здійснюється. У наведеному нижче прикладі `p` вказує на неіснуючий елемент:

```
int a[5] = {1, 2, 4, 8, 16};
```

```
int *p3 = a + 5;            // Такого елемента немає
```

Розіменування `p3` є небезпечним.

Вказівники можна використовувати для опису параметра функції типу масиву. Вихідний масив може бути змінений всередині функції:

```
void modifyStartingElement(int *p)
{
    p[0] = 0;
}

void main()
{
    int a[] = {1, 2, 3};
    modifyStartingElement(a);
    for (int i = 0; i < 3; i++)
    {
        cout << a[i] << ' '; // 0 2 3
    }
}
```

7. Використання динамічної пам’яті

При запуску програми операційна система налаштовує різні області пам’яті відповідно до вимог компілятора:

– глобальні змінні знаходяться у **глобальному просторі імен**, для якого виділяється спеціальний сегмент пам’яті.

– **регістри** утворюють особливу область пам’яті, вбудовану в центральний процесор.

– **стек** (stack, стек викликів) – це спеціальна область пам’яті, виділена для зберігання даних окремих функцій. Ознакою стеків як структур даних є принцип **last-in, first-out** (LIFO, першим зайшов, останнім вийшов).

– решта пам’яті, розподіленої для програми, – це так звана **динамічна пам’ять** (free store).

Програмісти можуть використовувати динамічну пам’ять для контрольованого виділення пам’яті й звільнення змінних із пам’яті. Оператор `new` використовують для розташування змінних у динамічній пам’яті.

Операція `new` повертає вказівник на об'єкт, розташований у динамічній пам'яті.

```
int *p = new int;
*p = 65;
. . .// *p можна вільно використовувати, поки пам'ять не буде звільнена
```

Змінна може бути ініціалізована початковим значенням:

```
int *p = new int(65); // *p = 65
```

Змінна, яка була створена у динамічній пам'яті, повинна бути звільнена за допомогою оператора `delete`:

```
delete p;
```

Після звільнення пам'яті, пам'ять, що раніше була виділена для змінної `*p` може бути використана для розташування інших змінних.

Якщо після імені типу в операції `new` розташувати квадратні дужки з цілим значенням всередині, у динамічній пам'яті можна розмістити масив відповідного типу. Ціле значення у квадратних дужках – це кількість елементів, для визначення якої можна використовувати будь-які вирази або змінні, що приводяться до `int`. Після створення використання динамічних масивів і звичайних масивів практично не відрізняються:

```
int n;
cin >> n;
double *pa = new double[n];
for (int i = 0; i < n; i++)
{
    pa[i] = 0;
}
. . .
```

Динамічну пам’ять, яка була виділена для масиву, звільняють у такий спосіб:

```
delete [] pa;
```

Необхідно стежити, щоб уся пам’ять, виділена за допомогою операції `new`, була звільнена за допомогою операції `delete`.

ПРИКЛАДИ ПРОГРАМ

1. Максимальний елемент. Наведена нижче програма знаходить максимальний елемент масиву цілих.

```
#include <iostream>
using namespace std;
void main()
{
    const int n = 4;
    int a[] = { 1, 2, 4, 8 };
    int indexOfMax = 0;
    for (int i = 1; i < n; i++)
    {
        if (a[i] > a[indexOfMax])
        {
            indexOfMax = i;
        }
    }
    cout << indexOfMax << ' ' << a[indexOfMax];
}
```

2. Сортування. Наведена нижче програма сортує елементи масиву в порядку зростання, використовуючи алгоритм сортування бульбашкою.

```
#include<iostream>
using namespace std;
void main()
{
    const int n = 5;
    double a[] = {11, 2.5, 4, 3, 5};
    bool mustSort; // повторюємо сортування
                   // якщо mustSort дорівнює true
```

```
    do
    {
        mustSort = false;
        for (int i = 0; i < n - 1; i++)
        {
            if (a[i] > a[i+1])
            // Обмінюємо елементи
            {
                double temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
                mustSort = true;
            }
        }
    }
    while (mustSort);
    for (int i = 0; i < n; i++)
    {
        cout << a[i] << ' ';
    }
}
```

3. Сума добутків. У наведеній нижче програмі визначено двовимірний масив і здійснюється знаходження суми добутків його рядків.

```
#include<iostream>
using namespace std;
void main()
{
    int a[][3] = {{1,2,3},
                  {2,3,4},
                  {0,1,2},
                  {1,1,12}};
    int sum = 0;
    for (int i = 0; i < 4; i++)
    {
        int product = 1;
        for (int j = 0; j < 3; j++)
        {
            product *= a[i][j];
        }
        sum += product;
    }
    cout << sum;
}
```

4. Заміна. У наведеній нижче програмі здійснюється заміна від'ємних елементів двовимірного масиву нулями.

```
#include<iostream>
using namespace std;
void main()
{
    double a[][3] = {{1,  -2,  3},
                     {2.1, 3, -4},
                     {0,-0.5, 11}};
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (a[i][j] < 0)
            {
                a[i][j] = 0;
            }
        }
    }
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << '\t' << a[i][j];
        }
        cout << endl;
    }
}
```

5. Сума елементів. У наведеній нижче програмі функція `sum()` обчислює і повертає суму елементів массиву.

```
#include <iostream>
using namespace std;
double sum(double *p, int n)
{
    double s = 0;
    for (int i = 0; i < n; i++)
    {
        s += p[i];
    }
    return s;
}

void main()
{
    double a[] = {1, 2, 3, 4};
    cout << sum(a, 4) << endl; // 10;
    cout << sum(a, 3) << endl; // 6;
```

```
}
```

6. Обчислення від'ємних елементів. Нехай задано масив дійсних чисел. Програма повинна обчислити кількість від'ємних елементів і створити масив відповідного розміру. Від'ємні значення повинні бути поміщені у новий масив.

Сирцевий код функції `main()` буде таким:

```
const int n = 7;
double a[n] = {10, 11, -3, -2, 0, -1, 4};
int count = 0;
int i;
for (i = 0; i < n; i++)
{
  if (a[i] < 0)
  {
    count++;
  }
}
double *b = new double [count]; // b зберігатиме від'ємні
елементи
int j = 0;                      // індекс у масиві b
for (i = 0; i < n; i++)
{
    if (a[i] < 0)
    {
        b[j] = a[i];
        j++;                    // наступний індекс
    }
}
for (j = 0; j < count; j++)
{
    cout <<  b[j] << ' ';       // виведення
}
delete [] b;                    // звільнення пам'яті
```

7. Двовимірний масив у динамічній пам'яті. Наступна програма читає кількість рядків і стовпців двовимірного масиву, розташовує масив цілих чисел у динамічній пам'яті, читає елементи масиву, обчислює суму початкових елементів рядків і звільняє пам'ять:

```
#include <iostream>
```

```
using namespace std;
void main()
{
    int m, n;
    cout << "Enter m and n:" << endl;
    cin >> m >> n;
    int i;

    // Створення масиву:
    int **a = new int* [m];
    for (i = 0; i < m; i++)
    {
        a[i] = new int [n];
    }

    // Уведення елементів:
    cout << "Enter elements of array:" << endl;
    for (i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> a[i][j];
        }
    }
    // Обчислення суми:
    int b = 0;
    for (i = 0; i < m; i++)
        b += a[i][0];
    cout << b;

    // Звільнення пам'яті:
    for (i = 0; i < m; i++)
    {
        delete [] a[i];
    }
    delete [] a;
}
```

ВПРАВИ ДЛЯ КОНТРОЛЮ

Завдання 1. Написати програму, яка обчислює суму додатних елементів масиву дійсних чисел.

Завдання 2. Написати програму, яка визначає масив дійсних чисел і обчислює суму елементів з непарними індексами.

Завдання 3. Написати програму, яка визначає масив цілих чисел і обчислює суму парних елементів.

Завдання 4. Написати програму, яка визначає двовимірний масив і обчислює добуток максимальних елементів його стовпців.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

1. Сума мінімального і максимального елементів

Написати програму, яка обчислює суму мінімального і максимального елементів масиву значень з рухомою крапкою подвійної точності. Здійснити пошук мінімального і максимального елементів у двох окремих функціях.

2. Сортування за зменшенням

Написати програму, яка сортує елементи масиву цілих чисел за зменшенням.

3. Сума додатних елементів

Написати програму, яка обчислює суму додатних елементів двовимірного масиву.

4. Масив у динамічній пам’яті

Написати програму, яка зчитує з клавіатури кількість рядків і стовпців двовимірного масиву, розташовує масив у динамічній пам’яті, зчитує елементи масиву з клавіатури, обчислює суми рядків і записує ці суми у новий масив.

5. Індивідуальне завдання

Створити програму, яка визначає та ініціалізує двовимірний массив цілих елементів, а потім реалізує такі дії:

– перетворення вихідного масиву відповідно до завдання, наведеного в колонці «Перший крок» табл. 5;

– створення у динамічній пам’яті та заповнення одновимірного масиву чисел типу `double` відповідно до завдання, наведеного у колонці «Другий крок» табл. 5;

– виведення на екран елементів обох масивів і звільнення пам’яті.

Треба передбачити виведення повідомлень про помилки, якщо перетворення або заповнення неможливі.

Таблиця 5 – Варіанти індивідуальних завдань

Номер варіанта (номер студента у списку)	Перший крок: правило перетворення елементів першого масиву	Другий крок: правило заповнення елементів другого масиву	Кількість рядків m	Кількість стовпців n
1	Усі елементи з непарними значеннями повинні бути збільшені у два рази	Квадратні корені мінімальних додатних елементів рядків	4	3
2	Усі елементи з парними значеннями повинні бути замінені їх квадратами	Кубічні корені мінімальних елементів колонок	3	5
3	Усі елементи з нульовим значенням слід замінити одиницями	Натуральні логарифми максимальних додатних елементів рядків	3	4
4	Усі елементи з парними значеннями повинні бути збільшені у два рази	Натуральні логарифми мінімальних додатних елементів колонок	4	5
5	Усі елементи повинні бути	Мінімальні елементи колонок	5	4

Продовження таблиці 5

Номер варіанта (номер студента у списку)	Перший крок: правило перетворення елементів першого масиву	Другий крок: правило заповнення елементів другого масиву	Кількість рядків *m*	Кількість стовпців *n*
	замінені їх абсолютними величинами			
6	Усі елементи з парними значеннями повинні бути збільшені в три рази	Кубічні корені діагональних елементів	3	3
7	Усі додатні елементи повинні бути замінені з цілими частинами їх десяткових логарифмів	Суми від’ємних елементів колонок	4	5
8	Усі від’ємні елементи повинні бути замінені їх квадратами	Квадратні корені діагональних елементів	4	4
9	Усі додатні елементи повинні бути замінені з цілими частинами їх натуральних логарифмів	Добутки від’ємних елементів рядків	5	4
10	Усі додатні елементи повинні бути замінені з	Максимальні додатні елементи рядків	3	5

Закінчення таблиці 5

Номер варіанта (номер студента у списку)	Перший крок: правило перетворення елементів першого масиву	Другий крок: правило заповнення елементів другого масиву	Кількість рядків *m*	Кількість стовпців *n*
	цілими частинами їх квадратних коренів			
11	Усі додатні елементи з парними значеннями повинні бути збільшені у два рази	Кубічні корені максимальних елементів колонок	5	4
12	Усі від'ємні елементи з непарними значеннями повинні бути збільшені у три рази	Квадратні корені мінімальних додатних елементів колонок	3	4
13	Усі негативні елементи з непарними значеннями повинні бути збільшені у два рази	Суми десяткових алгоритмів додатних елементів рядків	4	3
14	Усі додатні елементи з парними значеннями повинні бути збільшені в три рази	Залишки від ділення максимальних додатних елементів колонок на їхні десяткові логарифми	3	5

Для виклику стандартних математичних функцій слід включити заголовний файл `cmath`. Деякі корисні функції описані у табл. 6.

Таблиця 6 – Стандартні математичні функції Java

Математична функція	Значення
`double sqrt(double x);`	Обчислює квадратний корінь додатних елементів
`double pow(double x, double y);`	Обчислює `x` у степені `y`
`double log(double x);`	Обчислює натуральний логарифм `x`
`double log10(double x);`	Обчислює десятковий логарифм `x`
`int abs(int x);`	Повертає абсолютне значення цілого числа
`double fabs(double x);`	Повертає абсолютне значення числа з рухомою крапкою
`double sin(double x);`	Обчислює синус `x`
`double cos(double x);`	Обчислює косинус `x`
`double atan(double x);`	Обчислює арктангенс `x`

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке масив?
2. Що таке оператор індексування?
3. Яким є стартовий індекс масиву?
4. Чи можна використовувати змінні для визначення довжини масиву?
5. Як змінити розмір масиву після створення?
6. Як додати новий елемент у кінець масиву?
7. Як здійснюється передача масивів у функції?
8. Як отримати довжину масиву?
9. Як скопіювати значення елементів з одного масиву в інший?
10. Як визначити кількість стовпців у двовимірному масиві?
11. Як ініціалізувати двовимірний масив?

12. Чи можна створити двовимірний масив з різною довжиною рядків?

13. Що таке вказівник?

14. Як отримати адресу змінної?

15. Як здійснюється розіменування?

16. Які є варіанти використання вказівників?

17. У чому різниця між постійним указівником і вказівником на постійній об’єкт?

18. Як отримати адресу масиву?

19. Який є взаємозв’язок між масивами і вказівниками?

20. Що таке адресна арифметика?

21. Які області комп’ютерної пам’яті використовують для розташування змінних?

22. Що таке динамічна пам’ять?

23. Як розташувати змінну в динамічній пам’яті?

24. Чому необхідно видаляти непотрібні змінні з динамічної пам’яті?

25. Як розташувати двовимірний масив у динамічній пам’яті?

Рекомендована література

1. Bjarne Stroustrup. The C++ Programming Language. Third Edition / Bjarne Stroustrup. – Addison-Wesley, 1997.

2. Stanley B. Lippman C++ Primer. Third Edition / Stanley B. Lippman, Josee Lajoie. – Addison-Wesley, 1988.

3. Deitel H. M. C++. How to Program. Third Edition / H. M. Deitel, P. J. Deitel. – Prentice Hall, 2001.

4. Голуб Б. М. C#. Концепція та синтаксис / Б. М. Голуб. – Львів: Видавничий центр ЛНУ ім. Івана Франка, 2006. – 136 с.

5. Грицюк Ю. І. Програмування мовою С++: навч. посібник / Ю. І. Грицюк, Т. Є. Рак. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 292 с.

Internet-джерела

1. The C++ Programming Language (Bjarne Stroustrup's homepage) // http://www2.research.att.com/~bs/C++.html

2. ISO/IEC 14882:2003 Programming languages - C++ (International Stamdard) // http://cs.nyu.edu/courses/summer12/CSCI-GA.2110-001/downloads/C++%20Standard%202003.pdf

3. The C++ Resources Network // http://www.cplusplus.com/

4. The C++ Tutorial // http://www.learncpp.com/

5. C++ – Вікіпідручник // http://uk.wikibooks.org/wiki/C++

6. C++ – Вікіпіедія // http://uk.wikipedia.org/wiki/C++

7. Корх О. Основи мови програмування С++ // http://korkholeh.googlepages.com/cppfund.pdf

Навчальне видання

Методичні вказівки

до виконання лабораторної роботи 4

за темою «Використання масивів і вказівників»

з курсу «Алгоритмізація та програмування. Частина 1»

для студентів спеціальності

122 «Комп’ютерні науки»

Укладачі:

ІВАНОВ Лев Вадимович

БІЛОВА Марія Олексіївна

Відповідальний за випуск М. Д. Годлевський

Роботу до видання рекомендував О. В.Горілий

План 2018 р., поз. 313

Підписано до друку 28.05.2019. Гарнітура Times New Roman.

Ум. друк, арк. 1,5.

Видавничий центр НТУ «ХПІ»,

вул. Кирпичова, 2, м.Харків-2, 61002

Свідоцтво про державну реєстрацію ДК № 3478 від 21.08.2017 р.

Самостійне електронне видання